Dessin Vectoriel

Kilian Cousein & Benjamin Tardieu

2009-2010

Remerciements

Nous tenons à remercier tout particulièrement :
- Notre tuteur : M. Meynard pour ses conseils et son soutien technique.
- Les différents éditeurs des logiciels qui nous ont servis pour le développement mais aussi durant l'écriture de ce rapport.
- La faculté des Sciences pour les nombreuses connaissances qu'elle nous a permis d'acquérir, bien évidemment transmises par les professeurs mais également par les échanges entre étudiants.

Glossaire

MVC Le modèle-vue-contrôleur organise une application interactive en trois modules séparés : un pour le modèle de l'application accompagné de sa représentation des données ainsi que la logique du métier, un second pour les vues qui présentent les données à l'utilisateur et recueillent ses entrées, et le troisième pour un contrôleur qui achemine les requêtes et les flux de contrôle.

Java C'est un langage de programmation orienté objet, développé par Sun Microsystems. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (jsp).

Swing C'est une bibliothèque graphique pour le langage de programmation Java, faisant partie du package Java Foundation Classes (JFC), inclus dans J2SE. Swing offre la possibilité de créer des interfaces graphiques identiques quel que soit le système d'exploitation sous-jacent. Il utilise le principe Modèle-Vue-Contrôleur (MVC, les composants Swing jouent en fait le rôle du contrôleur au sens du MVC) et dispose de plusieurs choix d'apparence (de vue) pour chacun des composants standards.

Image vectorielle Une image vectorielle en informatique, est une image numérique composée d'objets géométriques individuels (segments de droite, polygones, arcs de cercle, etc.) définis chacun par divers attributs de forme, de position, de couleur, etc. Elle se différencie de cette manière des images matricielles (ou « bitmap »), dans lesquelles on travaille sur des pixels. L'intérêt est de pouvoir redimensionner l'image à volonté sans aucun effet d'escalier. L'inconvénient est que pour atteindre une qualité photo réaliste, il faut pouvoir disposer d'une puissance de calcul importante et de beaucoup de mémoire.

Bibliothèque / Librairie En informatique, une bibliothèque ou librairie est un ensemble de fonctions utilitaires, regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire. Les fonctions

sont regroupées de par leur appartenance à un même domaine conceptuel (mathématique, graphique, tris, etc).

Table des matières

Chapitre 1

Introduction

1.1 Généralités

Nous sommes amenés, dans le cadre de notre troisième année en Licence informatique à la faculté des Sciences de Montpellier, à travailler sur l'élaboration complète d'un projet, de son analyse à la conception puis à sa programmation. Tout au long du sixième semestre, le projet nous permet de comprendre quelles sont les phases de développement d'un projet et comment celui-ci doit être conduit. Nous pouvons par ailleurs mesurer notre capacité à réagir face à des problèmes en nous impliquant dans ce projet. Le projet que nous avons choisi est de réaliser un logiciel de Dessin Vectoriel.

Notre projet est né d'un problème existant. Comment faire pour pouvoir agrandir et rétrécir une image affichée à l'écran sans perdre de qualité ? La réponse : les images vectorielles*. Les images vectorielles ont la particularité d'avoir une résolution quasiment infinie car elles ne fonctionnent pas avec des pixels mais avec des formules mathématiques. Les formes géométriques qui composent l'image sont recalculées par la machine quand l'image est agrandie ou rétrécie.

Nous développerons en Java* pour nous permettre d'utiliser la librairie* graphique Swing* de J2SE[1] et de développer selon le principe de la programmation par objet. Ce projet sera réalisé tout au long de ce sixième semestre, en binôme sous la tutelle de Mr Michel Meynard.

1.2 Le sujet

Il s'agit d'écrire une bibliothèque* de composants logiciels permettant de :

- dessiner des dessins vectoriels et de les manipuler : ouvrir, sauver, importer, exporter,...
- les dessins pourront être de différents types
- stocker ces dessins

1. Java 2 Standard Edition

- les réduire ou les agrandir proportionnellement
- les afficher et les imprimer
- éventuellement les animer

1.3 Cahier des charges

Fonctionnnalités obligatoires

Manipulation de fichiers :

- ouvrir, enregistrer, importer, exporter des fichiers
- les dessins doivent pouvoir être de différents types
- possibilité de stocker ces dessins sous différents formats (PDF [2], PNG [3], SVG [4], ...)

Création d'objet :

Dessin :
- dessin de lignes à main levée et de segments de droite.

Forme :
- rectangles, carrés
- ellipses, cercles
- polygones

Outils de texte :
- police
- taille
- gras, souligné

Manipulation d'objet :
- transformations affines (translation, symétrie, redimensionnement)
- afficher et imprimer
- zoomer
- effacer

Contrôle des objets :
- revenir en arrière (annuler)
- revenir en avant (rétablir)

2. Portable Document Format
3. Portable Network Graphics
4. Scalable Vector Graphics

Fonctionnalités secondaires

- transformations affines (rotation, déformation)
- clones (copie d'un objet)
- les effets spéciaux : extrusion, effet miroir, dégradé de formes, morphage, ...
- animation des objets

Analyse des principaux logiciels de dessin vectoriel

Inkscape

Inkscape est un logiciel libre d'édition de graphismes vectoriels, utilisant le format de fichiers Scalable Vector Graphics (SVG) standard du W3C [5]. Les fonctionnalités supportées du format SVG incluent les formes, les chemins, le texte, les marqueurs, les clônes, les canaux alpha, les transformations, les motifs et les groupements. Inkscape supporte également les méta-données Creative Commons, l'édition de noeuds, les couches, les opérations de chemins complexes, la vectorisation des bitmaps, le texte suivant des chemins, le texte contournant des objets, l'édition XML [6] directe, ...Inkscape permet d'importer des formats tels que le Postscript, EPS [7], JPEG [8], PNG et TIFF [9], et exporte en PNG ainsi qu'en de nombreux formats vectoriels. Le but principal d'Inkscape est de créer un outil de dessin puissant et simple d'utilisation, totalement conforme aux standards XML, SVG et CSS [10].

Illustrator

Adobe Illustrator est le logiciel de création graphique vectorielle de référence dans les environnements professionnels. Il fait partie de la gamme Adobe et peut être utilisé indépendamment ou en complément de Photoshop, offre des outils de dessin vectoriel puissants. Les images vectorielles sont constituées de courbes générées par des formules mathématiques. L'un des outils principaux d'Illustrator étant « la plume » qui permet de tracer des courbes à l'aspect parfait grâce au placement de points d'ancrage et de tangentes qui vont en modifier la courbure. Adapté aussi bien à la création de document papier qu'à celle d'illustrations pour Internet (logos, affiches, etc.) ce logiciel est orienté vers le marché professionnel, il intègre de nombreuses options propres à améliorer la productivité.

5. World Wide Web Consortium
6. Extensible Markup Language
7. Encapsulated PostScript
8. Joint Photographic Experts Group
9. Tagged Image File Format
10. Cascading Style Sheets

Microsoft Expression Design

Microsoft Expression Design permet de réaliser des graphismes de grande qualité pour des applications internet ou de bureau. L'application est intégralement compatible avec Expression Blend, et permet d'exporter facilement vos graphiques via XAML. Ce qui permet d'animer facilement vos créations graphiques. Les effets utilisés par le logiciel sont intégralement non destructifs, c'est-à-dire qu'il est toujours possible de retrouver le graphique de départ après l'application d'un effet. En utilisant la puissance du dessin vectoriel et la facilité de prise en main de l'interface, le logiciel permet de créer des images de qualité simplement.

La syntaxe

Lors du développement, nous coderons en « français » c'est à dire que les attributs, les noms de classes, les méthodes, et les fonctions que nous utiliserons pour créer cet éditeur de dessin vectoriel auront des noms en français.

Classe : majuscule puis minuscule et majuscule pour séparer les mots. (exemple : `MaClasse`)

Méthode : minuscule et majuscule pour séparer les mots. (exemple : `maMethode`)

Attribut : minuscule et majuscule pour séparer les mots. (exemple : `monAttribut`)

Indentation : Mettre le maximum d'accolades. Et les placer en dessous du for et de la dernière ligne de la boucle.

Commentaires : les mettre au-dessus de chaque fonction, de chaque classe. Eclipse sera utilisé pour générer la documentation. Gestion des erreurs avec le format : Erreur : courte description de l'erreur, situées sur nom de la page, a telle ligne. Les commentaires seront de la forme suivante pour :

– Les classes :
```
/**
  * Description de la classe.
  *
  * @author Kilian Cousein et Benjamin Tardieu
  */
```

– Les variables :
```
TypeVariable nomVariable ; // description de la variable
```

– Les méthodes :

```
/**
  * Description de la methode.
  *
  * @param nomParametre Description du paramètre.
  * @return description de la valeur retournée par la méthode.
  */
```

Chapitre 2

Organisation du projet

2.1 Organisation du travail

Les membres de l'équipe du projet

L'équipe de travail chargée du projet est composée de deux membres :
- Kilian Cousein `<kilian.cousein@etud.univ-montp2.fr>`
- Benjamin Tardieu `<benjamin.tardieu@etud.univ-montp2.fr>`

L'équipe ne comportant que deux membres, ceux-ci se sont vu attribuer le titre de Chef de projet.

La communication

Au sein de l'équipe de projet, nous avons mis en place deux moyens de communication.

A distance

Pour communiquer à distance, nous utilisons les services en ligne de Google. Ainsi, nous disposons d'un wiki, et d'un dépôt SVN pour pouvoir mettre à jour le programme sans conflits et déposer régulièrement une nouvelle version du programme réalisé

En réunion

Nous organisons une réunion au moins une fois par semaine pour faire le point sur le projet. On discute de l'avancée du projet, on pose des questions, on obtient des réponses, on donne des suggestions. Tout cela dans le but de cerner les problèmes auxquels on est confronté pour les résoudre ensuite en groupe.

Décomposition du travail

Voici la répartition des tâches qu'a réalisé chaque étudiant :

- Kilian Cousein :
  Kilian COUSEIN a travaillé sur l'Interface Homme Machine (IHM). Il a construit l'IHM, et quelques motifs (Carré, Ellipse, Ligne, texte).
- Benjamin Tardieu :
  Benjamin TARDIEU s'est occupé des fonctionnalités suivantes : implémentation de modèle MVC, gestion de l'historique, gestion d'un fichier et le motif polygone.

2.2 Choix des outils de développement

Langage

Le langage Java nous a été imposé ce qui nous correspond très bien car :

- Java possède un ramasse-miettes qui simplifie le travail du programmeur (il n'a plus besoin de s'occuper de la gestion de la mémoire et d'écrire des destructeurs pour ses classes).
- Java possède une machine virtuelle qui permet une compatibilité du programme avec un grand nombre de plateformes sans avoir à recompiler le code source.

Aussi, il est à noter que l'API [1] Java inclut la bibliothèque « Swing » qui nous sera très utile pour le développement de l'interface graphique.

Editeur

Nous avons choisi Eclipse comme plateforme de développement car celui-ci est entièrement paramétrable est facile d'utilisation. Il permet l'auto-complétion, la coloration syntaxique du texte, la visualisation de la hiérarchie des attributs et des méthodes dans les classes et surtout une bonne gestion des gestionnaires de versions SVN.

Compilateur

Nous avons choisi javac, le compilateur, qui convertit le code source en fichier .class (contenant le bytecode Java).

Débugueur

Jdb est le débogueur de base intégré à l'IDE [2] éclipse. Celui-ci permet de débogué du code source Java.

1. interface de programmation
2. Environnement de développement intégré

Gestionnaire de Versions

Google SVN nous a permis de mettre en commun de nos travaux sans difficulté, ainsi nous avons pu développer le projet paralellement (`http://code.google.com/p/dvectoriel/`).

Outils d'analyse

Umbrello est un outil de développement qui nous a permis de créer des diagrammes UML [3] et de générer le code source correspondant pour commencer la partie développement.

3. Langage de modélisation unifié

Chapitre 3

Analyse du projet

Avant de commencer à programmer nous avons dû réaliser une structure de base sur laquelle nous avons ensuite travaillé. Nous nous sommes donc réunis pour discuter de la manière dont le logiciel devait être développé (Quelle classe ? Comment les nommés ?...)

Nous nous sommes servis du logiciel Umbrello pour créer des diagrammes de classes qui nous ont par la suite permis de générer du code Java et de commencer à développer par la suite (après répartition des taches de travail).

Il faut tout d'abord savoir que nous utilisons le modèle MVC [1]. L'architecture de notre application suivra donc ce modèle. Nous avons donc séparé la vue (ou IHM [2]) et le modèle qui seront totalement indépendants. Le contrôleur permettra de faire le lien entre la vue et le modèle. Nous allons maintenant vous présentez comment nous avons conçu ces 3 parties du programme.

Il faut savoir que la vue, le modèle et le contrôleur seront représentés par des packages.

3.1 Le modèle

Celui-ci doit représenter ce qu'est un dessin vectoriel. Nous avons conçu le dessin de la manière suivante : Une classe Dessin possédant un vecteur de motifs.

Les motifs sont les formes dont est composée l'image vectorielle. Ce sont des cercles, des lignes, des carrés, des rectangles, des polygones, etc ayant chacun leurs propres caractéristiques : un point de coordonnée (x, y), une hauteur, une largeur, une couleur... Le modèle va donc contenir une classe abstraite Motif. Celle-ci doit être abstraite car un Motif ne peut être instancié, les seuls objets pouvant être instanciés seront les classes Rectangle, Ellipse, Polygone, Ligne... qui elles hériteront de la classe Motif.

1. Modèle-Vue-Contrôleur*
2. Interface Homme Machine

Lorsqu'on a commencé à remplir le contenu des classes on s'est aperçu que le modèle n'était pas complet. En effet, parmi tous les motifs, on a distingué des propriétés que possédaient certains objets et pas d'autres. Par exemple, les ellipses et les rectangles possèdent chacun une hauteur, une largeur et un point (x, y) qui va permettre à un objet de la librairie Swing de dessiner l'objet à l'écran. Mais un objet de type ligne possède deux points (x1, y1) (x2, y2) et pas de hauteur ni de largeur. Il a donc fallu modifier la classe Motif pour que celle-ci soit plus générale. On a retiré les attributs hauteur et largeur et les méthodes correspondantes et nous les avons placé dans une nouvelle classe abstraite MotifBordure qui hérite de Motif. Ainsi, certains motifs héritent directement de la classe Motif alors que d'autres en héritent indirectement par le biais de la classe MotifBordure.

Pour la gestion des évènements (annuler ou rétablir une modification/-création/suppression d'objet), nous avons créé une classe Historique. Celle-ci contiendra deux piles, une pour la gestion des évènements à annuler et une pour la gestion des évènements à rétablir. Le dessin possèdera une instance de cette classe.

Les évènements quant à eux suivront le principe du pattern état. Nous aurons donc la classe abstraite Evenement qui représentera un évènement. Les classes CreationEvenement, ModificationEvenement et SuppressionEvenement héritent d'Evènement, elles représenteront l'état d'un évènement.

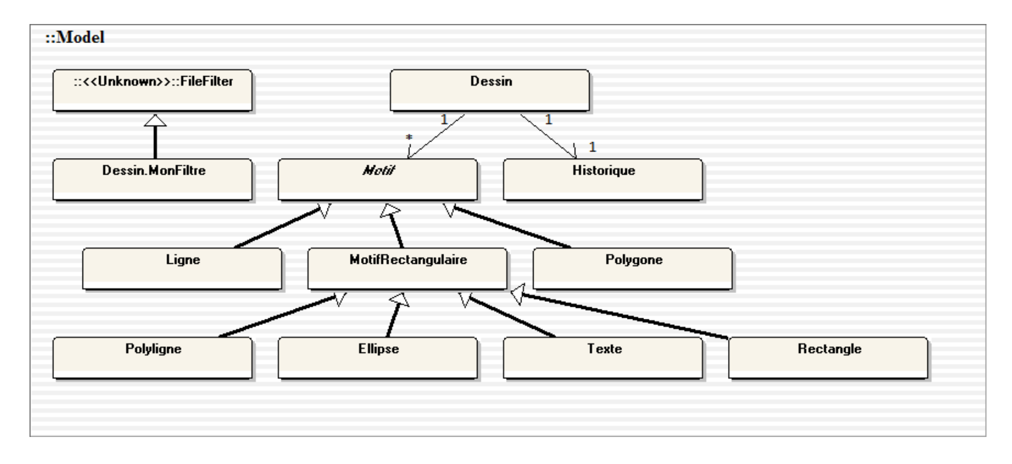


FIGURE 3.1 – Diagramme UML du Modèle

3.2 La vue

Celle-ci va contenir toutes les classes représentant les objets qui composeront la fenêtre, y compris la fenêtre elle-même. Ces objets seront tous des composants Swing de Java et ils hériteront donc de leur classe correspondante. Il y aura donc, un menu principal (avec fichier, nouveau, ouvrir,

édition, etc.. . .), une barre d'outils verticale avec a peu près les mêmes fonctions que le menu principal, une autre barre verticale qui servira à changer les propriétés des objets (couleur, opacité, police pour le texte. . .), une barre horizontale sur le côté gauche de la fenêtre qui servira à choisir les objets à dessiner, une zone de dessin dans laquelle sera dessinée l'image vectorielle, une autre barre horizontale sous la zone de dessin pour pouvoir modifier la couleur de l'objet et de sa bordure et afficher les coordonnées de la souris dans la zone de dessin.

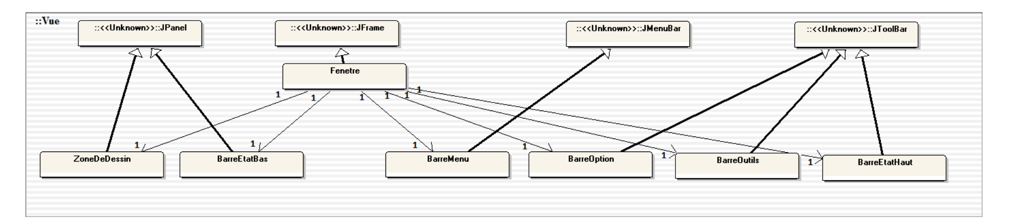


FIGURE 3.2 – Diagramme UML de la vue

3.3 Le contrôleur

Celui-ci sera représenté par une classe Controleur et va posséder toutes les méthodes permettant de faire la liaison entre la vue et le modèle. C'est lui qui va instancier la vue et le modèle qui seront respectivement représentés par un objet fenêtre et un objet dessin.

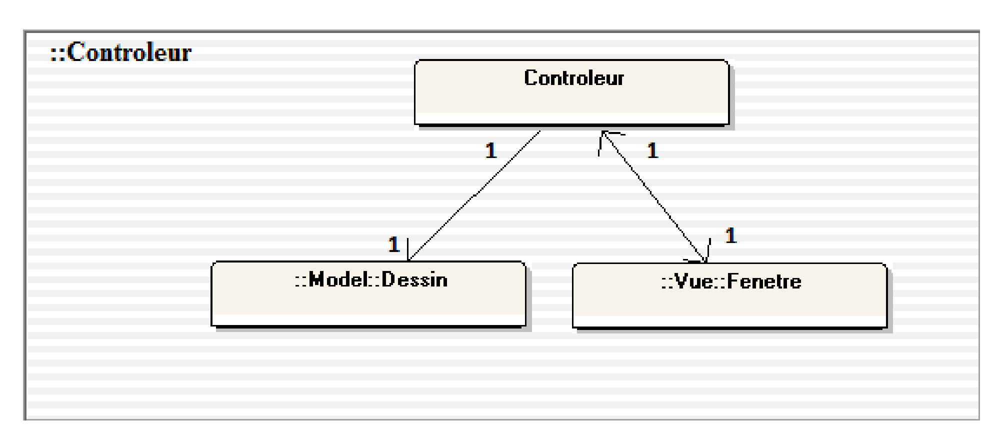


FIGURE 3.3 – Diagramme UML du contrôleur

Chapitre 4

Développement

Apres avoir terminé l'analyse du projet, le développement du projet a pu commencer. Celui-ci est découpé en trois partite en relation avec le model MVC vu précédemment. Ces trois parties sont donc le développement du model, de la vue et du contrôleur. Pour chacune de ces parties nous expliquerons le fonctionnement global des classes correspondantes. Nous n'expliquerons pas toutes les méthodes en détails car elles sont commentées dans le code du programme. Nous pensons qu'il n'est pas nécessaire de répéter les explications.

4.1 Le modèle

Le modèle est divisé en deux parties. La première qui représente la modélisation du dessin vectoriel et la deuxième qui représente la modélisation des évènements.

Modélisation du dessin vectoriel

Cet ensemble de classes représente la décomposition du problème de la modélisation d'un dessin vectoriel.

Classe Dessin

Un dessin est constitué d'un ensemble de variables qui permettent une modélisation la plus proche possible de la réalité. Ces variables sont :

- String url : Une chaine de caractères qui permet de stocker l'adresse du fichier. Celle-ci est vide si aucun fichier n'a été ouvert ou si le dessin en cours n'a pas étais sauvegardé.
- int largeur, hauteur : Dimension (largeur et hauteur) du dessin défini par defaut en 800*600, hauteur : Dimension (largeur et hauteur) du dessin défini par default en 800*600.

- Vector<Motif> motifs : vecteur qui va contenir l'ensemble des motifs constituant le dessin en cours.
- float zoom : nombre réel représentant la valeur du zoom qui sera utilisé pour recalculer les tailles et coordonnées du dessin et des motifs concernés.
- boolean enregistrer : ce booléen est vrai si le dessin est enregistré, faux sinon.
- Historique historique : représente l'historique des évènements générés par l'utilisateur. La classe historique sera détaillée plus bas.
- Motif motif : Elle contient le motif en cours de création ou le motif sélectionné par l'utilisateur ou encore un motif avec toutes ses caractéristiques à nul si l'utilisateur a cliqué sur un type de motif à dessiner et qu'il n'a pas encore tracé la figure. Puis si l'utilisateur clique dans un espace vide de la zone de dessin, elle sera égale à nulle.
- Color couleur : cette variable est initialisée à la couleur bleu. Elle contient la couleur sélectionnée par l'utilisateur dans l'outil de sélection de couleur ou la couleur du dessin sélectionné s'il y en a un. Le type Color stocke la couleur au format RGB [1].
- Color couleurBordure : initialisée à la couleur noir. Elle contient la couleur sélectionnée par l'utilisateur dans l'outil de sélection de couleur pour les bordures ou la couleur de la bordure du dessin sélectionné s'il y en a un.

Cette classe contient les méthodes pour :
- enregistrer, ouvrir, faire un nouveau dessin.
- dessiner le dessin grâce à un composant de la classe Graphics de la librairie AWT [2] de java.
- modifier le dessin en cours de création (ajouter, créer, modifier, supprimer, trouver un motif).

Classe Motif

Un motif est représenté par un ensemble de variables :
- nt x, y : ces deux entiers forment un couple qui représente le point (x, y). Ce point représente le coté le plus en haut à gauche du motif.
- int largeurBordure : est égale a zéro si le motif ne possède pas de largeur sinon elle est égale à la valeur de la largeur de la bordure.
- Color couleurBordure : la couleur de la bordure du dessin.
- Color couleur : La couleur du dessin.
- float opacite : Valeur de l'opacité du motif (entre 0 et 1)
- float opaciteBordure : Valeur de l'opacité de la bordure du motif (entre 0 et 1) Pour l'opacité plus on est proche de 0 plus l'objet concerné est

1. Rouge Vert Bleu
2. Abstract Window Toolkit

transparent. Plus on est proche de 1, moins l'objet est transparent.
– boolean selectionne : est vrai si l'utilisateur a sélectionné le motif, faux sinon.
– int poigneSelectionne : Lorsque l'utilisateur a sélectionné un motif et que celui-ci à cliqué sur l'une des poignées autour du motif pour le redimensionner, cette variable va contenir le numéro de la poignée sur lequel l'utilisateur a cliqué.
– static final int COTE_POIGNEE : Il y a 9 variables de cette forme. « COTE » représente l'endroit du motif ou est situé la poignée comme par exemple pour en haut à gauche : GAUCHE_HAUT_HANDLE.
– static final int TAILLE_POIGNEE : représente la longueur des côtés d'une poignée (qui est un carré). Pour notre application on fixera la taille d'une poignée a 9 pixels.

Cette classe est abstraite donc on ne peut instancier un motif. Cela n'aurait aucun sens de dessiner un objet Motif car ce serait seulement un point avec une couleur. Grace à l'abstraction de la classe on va pouvoir dire quelles méthodes vont devoir être redéfinir pour pouvoir ensuite modéliser un vrai motif comme un rectangle, etc.

Ces méthodes abstraites sont donc essentielles pour les motifs que l'on va dessiner. C'est elles qui vont donner forme à l'objet, c'est pour cela qu'il faut les redéfinir. Ces méthodes servent donc à :
– dessiner le motif et ses poignées
– le déplacer, le redimensionner
– le copier
– l'ouvrir et l'enregistrer
Le reste des méthodes seront donc des accesseurs qui permettront de modifier les variables citées ci-dessus, essentielles pour modifier les propriétés du motif.

Pour l'enregistrement d'un Motif, il faudra redéfinir la méthode enregistrer dans la sous classe correspondante. Un motif au format SVG à la forme suivante :

```
<object fill="x="170" y="287"  height="46" width="44"
rgb(0,255,255)"  stroke="rgb(0,0,0)" stroke-width="1"
opacity="0.42" opacity-stoke="0.42" />
```

La balise object permet de dire que l'objet à dessiner est de type object, elle est fermé grâce à « /> ».
– x représente le l'abscisse x du motif.
– y représente l'ordonne y du motif.
– height représente la hauteur.

- width représente la largeur.
- rgb(0,255,255) représente la couleur du motif au format RGB(rouge, vert, bleu).
- stroke="rgb(0,0,0)" représente la couleur de la bordure du motif
- stroke-width="1" représente la largeur de la bordure. Si elle est égale à 0 alors il n'y a pas de bordure.
- opacity="0.42" représente la transparence du motif.
- opacity-stroke="0.42" représente la transparence de la bordure.

Pour ouvrir un motif il s'agira d'associer chaque valeur de chaque attribut de la balise à sa variable correspondant dans la classe.

Classe MotifRectangulaire

Cette classe hérite de la classe Motif et est abstraite elle aussi. Elle permet de spécifier encore plus les types de motifs que l'on pourra dessiner. En effet, on a distingué deux sortes de motifs :
- les motifs qui vont se dessiner grâce à un point(x, y) et à leur largeur et leur hauteur
- les motifs un peu plus complexes qui nécessitent d'autres caractéristiques comme par exemple le polygone qui nécessite autant de point (Xi, Yi) que de sommets qu'il possède.

Cette classe représentera donc les motifs assez simple qui nécessitent seulement un point(x, y), une largeur et une hauteur pour être dessinés. Nous avons donc nommé cette classe en fonction de ces caractéristiques.

Celle-ci a donc deux attributs supplémentaires :
- int largeur : représente la largeur du motif à dessiner
- int hauteur : représente la hauteur du motif à dessiner

Ensuite quelques méthodes de la classe abstraites Motif ont était redéfinis. En effet les objets de cette classe ayant des caractéristiques en commun, ils vont donc posséder des méthodes en commun. Ces méthodes servent à :
- dessiner les poignées
- déplacer, redimensionner

En effet, ces méthodes utilisent le point(x, y) du motif, la largeur et la hauteur ainsi que d'autres attributs de la classe Motif. Donc en les définissant dans cette classe, nous n'aurons pas besoin de réécrire des copies de ces méthodes pour chaque classe qui héritera de MotifRectangulaire.

Classe Rectangle

Cette classe hérite de MotifRectangulaire et donc indirectement de Motif. Comme son nom l'indique, elle représente un objet de type rectangle.

Les méthodes qui n'ont pas été redéfini dans MotifRectangulaire vont être redéfinir ici. Ces méthodes servent à :

- dessiner le motif
- copier
- ouvrir, enregistrer

Ces méthodes sont propres a l'objet, c'est pour ça qu'elles sont redéfinies ici.

Pour dessiner un rectangle on utilisera l'objet Graphics de la librairie AWT de Java. La méthode `fillRect(int x, int y, int largeur, int hauteur)` permettra d'y remédier.

Pour enregistrer un rectangle il s'agit de le mettre sous sa forme SVG. En svg un rectangle est représenté de la manière suivante dans le fichier :

```
<rect />
```

Pour voir les attributs de la balise voir la classe Motif. Pour ouvrir un rectangle il suffira de récupérer les attributs de la balise correspondante.

Classe Ellipse

Cette classe hérite de MotifRectangulaire et donc indirectement de Motif. Comme son nom l'indique, elle représente un objet de type ellipse. Les méthodes qui n'ont pas été redéfinies dans MotifRectangulaire vont être redéfinir ici. Ces méthodes servent à :

- dessiner le motif
- copier
- ouvrir, enregistrer

Ces méthodes sont propres a l'objet, c'est pour ça qu'elles sont redéfinies ici.

Pour dessiner une ellipse on utilisera l'objet Graphics de la librairie AWT de Java. La méthode `fillOval(int x, int y, int largeur, int hauteur)` permettra d'y remédier.

Pour enregistrer une ellipse il s'agit de la mettre sous sa forme SVG. En svg une ellipse est représentée de la manière suivante dans le fichier :

```
<ellipse cx="128" cy="108" rx="53" ry="31" />
```

Les attributs :

- cx="128" représente l'abscisse x du centre de l'ellipse.
- cy="108" représente l'ordonné y du centre de l'ellipse
- rx="108" représente le rayon sur l'axe des abscisse
- ry="108" représente le rayon sur l'axe des ordonnés

Pour voir les autres attributs de la balise voir la classe Motif.

Pour ouvrir une ellipse il s'agira donc de récupérer les attributs ci-dessus et de recalculer le point (x, y) ainsi que la largeur et la hauteur. Pour :
- x = cx –rx ;
- y = cy – ry ;
- largeur = rx * 2 ;
- hauteur = ry * 2 ;

Classe Texte

Cette classe hérite de MotifRectangulaire et donc indirectement de Motif. Comme son nom l'indique, elle représente un objet de type texte. Quelques attributs supplémentaires permettent de définir un objet de type texte.
- String police = "Verdana" ; C'est la valeur de la police du texte. Elle est initialisée à Verdana.
- int taillePolice = 14 ; C'est la taille du texte en pixel. Elle est initialisée à 14.
- String texte ; C'est la valeur du texte que l'utilisateur a rentré.

Les méthodes qui n'ont pas été redéfinies dans MotifRectangulaire vont être redéfinir ici. Ces méthodes servent à :
- dessiner le motif
- copier
- ouvrir, enregistrer

Ces méthodes sont propres a l'objet, c'est pour ça qu'elles sont redéfinies ici.

Pour dessiner du texte on utilisera l'objet Graphics de la librairie AWT de Java. La méthode `drawString(int x, int y, string texte)` permettra d'y remédier.

Pour enregistrer du texte il s'agit de le mettre sous sa forme SVG. En svg le texte est représenté de la manière suivante dans le fichier :

```
<text font-family="Verdana" font-size="10" />
```

Pour voir les attributs de la balise voir la classe Motif.

Les attributs :
- font-family="Verdana" représente la police du texte à afficher
- font-size="10" représente la taille du texte en pixel

Pour ouvrir du texte il s'agira d'associer les attributs de la balise text avec leurs variables correspondantes.

Classe Polygone

Cette classe hérite de Motif. Comme son nom l'indique, elle représente un objet de type polygone. Un polygone est représenté par un ensemble de points (xi, yi) et son nombre de points. Des attributs supplémentaires sont donc nécessaires pour le définir :

- int pointX[] : tableau des abscisse xi du polygone.
- int pointY[] : tableau des ordonnés yi du polygone.
- int nbPoint : nombre de point du polygone.

Les méthodes abstraites de la classe Motif vont devoir être redéfinies en fonction de ses variables supplémentaires. Une méthode supplémentaire va être ajoutée. C'est la méthode `calculerPolygone()`. Celle-ci va permettre de calculer l'ensemble des points du polygone en fonction d'un point central défini par l'utilisateur.
Les deux formules utilisées sont :

$(\text{rayon } \times \cos(((i \times \pi) + decalage) \div (nbPoint \div 2))) + x$

$(\text{rayon } \times \sin(((i \times \pi) + decalage) \div (nbPoint \div 2))) + y$

La première permet de calculer la coordonnée xi et la deuxième la coordonnée yi des points du polygone.

Le rayon est la distance du centre à n'importe quel point du polygone.

i est le numero d'un des points du polygone qui varit entre 0 et nbPoint - 1.

nbPoint représente le nombre de point que possède le polygone.

Le décalage sert seulement à faire tourner la figure sur elle-même. x et y représentent respectivement l'ordonnée et l'abscisse du centre du polygone, c'est le point où a cliqué l'utilisateur pour dessiner sa figure.

Pour dessiner un polygone on utilisera l'objet Graphics de la librairie AWT de Java. La méthode `fillPolygon(int[] pointX, int[] pointY, int nbPoint)` permettra d'y remédier.

Pour enregistrer un polygone il s'agit de le mettre sous sa forme SVG. En svg un polygone est représenté de la manière suivante dans le fichier :

```
<polygon points="213,437 192,408 207,376 241,373 262,402 247,434
"/>
```

L'attribut « points » représente l'ensemble des point (xi, yi). Chaque point est séparé par un espace. Les abscisses et les ordonnés sont séparés par une virgule.

Pour voir les autres attributs de la balise voir la classe Motif. Pour ouvrir un polygone il suffira de récupérer l'attribut « points » et de parser son contenu pour affecter les variables correspondantes.

Classe Ligne

Cette classe hérite de Motif. Comme son nom l'indique, elle représente un objet de type ligne. Une ligne est représentée par deux points (x1, y1), (x2, y2). La classe Motif nous donne déjà le point(x1, y1) nommé (x, y) dans la classe Motif. Quant au point (x2, y2) celui-ci va être défini dans la classe Ligne de la manière suivante :

- int x2
- int y2

Les méthodes abstraites de la classe Motif vont devoir être redéfinies en fonction de ses variables supplémentaires.

Pour dessiner une ligne on utilisera l'objet Graphics de la librairie AWT de Java. La méthode `drawLine(int x1, int y1, int x2, int y2)` permettra d'y remédier.

Pour enregistrer une ligne il s'agit de la mettre sous sa forme SVG. En SVG une ligne est représentée de la manière suivante dans le fichier :

```
<line x1="102" y1="73" x2="56" y2="83"/>
```

Les attributs x1, y1, x2, y2 représentent respectivement les points x, y, x2, y2.

Pour ouvrir une ligne il suffira de récupérer les attributs correspondant de la balise et d'affecter leurs valeurs aux variables correspondantes.

Classe Historique

Celle-ci va permettre à la classe Dessin de gérer deux piles d'évènements.

Ces piles d'évènements vont permettre de gérer les actions (ou évènement) produites par l'utilisateur utilisant le programme. Elles serviront à rétablir ou annuler une action lorsque l'utilisateur cliquera sur le bouton correspondant. Elles sont représentées de la manière suivante :

- Stack<Evenement> historiqueAnnuler : Cette pile contiendra tous les évènements produits par l'utilisateur (création, redimensionnement, déplacement, effacement d'un motif).
- Stack<Evenement> historiqueRetablir : Cette pile permettra de garder en mémoire les évènements annulés par l'utilisateur pour pouvoir ensuite les rétablir au besoin.

Les méthodes de cette classe permettront d'empiler ou dépiler les piles d'évènements pour pouvoir gérer efficacement la gestion des évènements.

Modélisation des évènements

Les évènements sont modélisés selon le pattern état. Nous aurons donc une classe abstraite Evènement pour modéliser un Evenement. Puis nous

aurons des sous-classes héritant d'Evenement qui représenteront l'état d'un évènement.

Nous avons distingué trois sortes d'évènements :

- Les évènements de type création qui permette de savoir si l'utilisateur à créer un motif.
- Les évènements de type suppression qui permettent de savoir si l'utilisateur a supprimer un motif.
- Les évènements de type modification qui permettent de savoir si l'utilisateur a déplacé ou redimensionné un motif.

Voici le détail des classes de ce package.

Classe Evenement

Un évènement est modélisé par trois variables :

- Motif motifInitial : c'est le motif qui a été modifié par l'utilisateur, sans la prise en compte des modifications.
- Motif motifApresEvenement : c'est le motif qui a été modifié par l'utilisateur, avec la prise en compte des modifications.
- Dessin dessin : c'est le dessin dans lequel le motif est dessiné.

Cette classe est abstraite, elle possède donc des méthodes abstraites qui devront être redéfinies dans les sous-classes héritées.

Ces méthodes permettront de :

- Rétablir un évènement : méthode retablir()
- Annuler un évènement : méthode annuler()

Le reste des méthodes seront des accesseurs permettant de modifier les variables de classes.

Pour les classes héritées décrites plus bas nous nous contenterons de décrire les méthodes abstraites et leur fonctionnement.

Classe CreationEvenement

Pour la méthode annuler(), il s'agit de supprimer du dessin le motif qui a été créé.

Pour la méthode retablir(), il s'agit d'ajouter au dessin le motif précédemment supprimé.

Classe SuppressionEvenement

Pour la méthode annuler(), il s'agit d'ajouter au dessin le motif qui a été précédemment supprimé par l'utilisateur.

Pour la méthode retablir(), il s'agit de supprimer du dessin le motif précédemment ajouté.

Classe ModificationEvenement

Pour la méthode annuler(), il s'agit de remplacer dans le dessin l'ancien motif par le nouveau motif modifié. Pour la méthode retablir(), il s'agit de remplacer dans le dessin le nouveau motif par l'ancien motif.

4.2 La vue

La difficulté de la vue ou Interface Homme Machine réside surtout dans l'ergonomie car grâce à la librairie Swing tous les composants graphiques sont déjà conçus. Ainsi, pas besoin de fabriquer une fenêtre, il suffit d'utiliser la classe JFrame, idem pour la barre d'outils avec la classe JToolBar, JMenuBar pour la barre de menu, le reste des barres seront des JPanel ainsi que la zone de dessin. Il faut savoir que chaque composant de l'IHM possède un objet contrôleur qui va permettre de faire la liaison avec le modèle en fonction des actions que l'utilisateur va faire sur la vue.

Classe Fenetre

Elle hérite de JFrame, ce qui permet d'afficher une fenêtre standard à l'écran. Celle-ci va contenir tous les composants graphiques qui vont être affichés à l'écran. Ces composants sont des variables :

– BarreMenu barreMenu ;
– BarreOption barre0ption ;
– BarreOutils barre0utils ;
– BarreEtatHaut barreEtatHaut ;
– BarreEtatBas barreEtatBas ;
– ZoneDeDessin zoneDeDessin ;

Celles-ci seront décrites ci-dessous.

Lors de la construction de la fenêtre, tous les composants vont être instanciés puis la méthode init() être appelée et va permettre de placer les composants dans la fenêtre. Les autres méthodes servent à faire la liaison entre les composants de la fenêtre et la fenêtre.

Classe BarreMenu

Elle possède deux menus (Fichier, Edition) qui sont des JMenu eux même composés de JMenuItem qui représentent les options des menus. Le menu Fichier permet d'utiliser les fonctions principales du programme. Le menu Edition permet pour l'instant d'annuler et rétablir des évènements générer par l'utilisateur, mais celui-ci sera complété à l'avenir avec diverses options (copier, couper, coller. . .).

Fichier Edition

Figure 4.1 – Barre de menu

Chaque JMenuItem possède un écouteur de la classe ActionListener qui permet à la machine de savoir sur quelle option l'utilisateur a cliqué.

Il faut savoir que pour la gestion des écouteurs la classe BarreMenu hérite de l'interface ActionListener. L'objet contrôleur va permettre d'agir sur le modèle en fonction de l'évènement déclenché. C'est donc la méthode actionPerformed(ActionEvent e) qui est une méthode héritée de l'interface ActionListener qui va être appelée lorsqu'un écouteur va être déclenché. L'objet ActionEvent permettra de savoir sur quelle option l'utilisateur a cliqué. Celle-ci va ensuite utiliser les méthodes du contrôleur pour exécuter l'évènement demandé.

Classe BarreOption

Celle-ci contient diverses options. On retrouve la majorité des options du menu, et d'autres relatives à la conception du dessin (Zoom-, Zoom+). Elle permet d'accéder plus rapidement aux options du menu et elle est surtout plus esthétique.

Cette classe hérite de JPanel, cela permet d'obtenir une zone vide dans laquelle on peut facilement disposer des objets. Elle hérite aussi de ActionListener, c'est le même principe que précédemment pour les écouteurs.

Les objets qui la composent sont tous des JButton composés d'une image contenu dans le dossier /icon du projet. Ils possèdent tous un ActionListener qui va permettre de gérer les évènements générés par l'utilisateur. Ce sera ensuite la méthode ActionListener(ActionEvent e) qui s'occupera de faire la liaison avec le modèle grâce au contrôleur.

Lorsque l'utilisateur cliquera sur l'un des boutons de sauvegarde ou de chargement. Un objet JFileChooser de Swing permettra d'ouvrir une fenêtre pour sélectionner l'emplacement du fichier à ouvrir ou enregistrer.



Figure 4.2 – Barre d'options

Classe BarreOutils

Elle contient tous les outils nécessaires au dessin de l'image vectorielle. Elle permet de créer, d'effacer, sélectionner des objets.

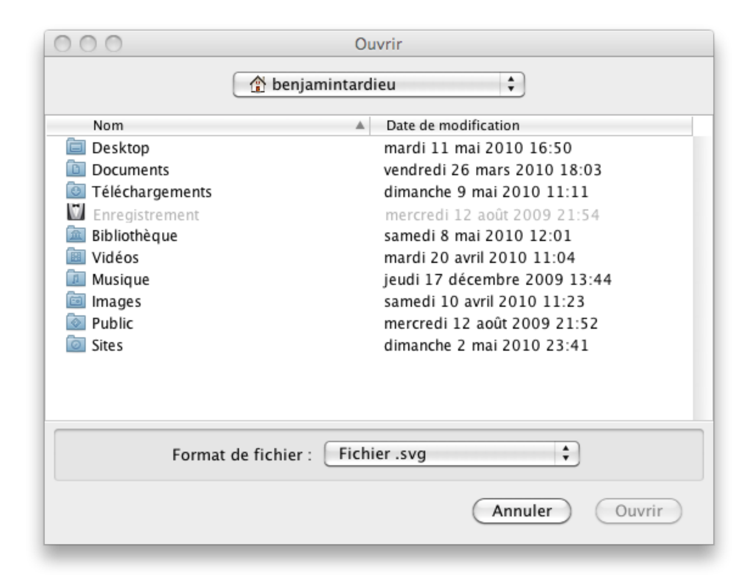


FIGURE 4.3 – Fenêtre Ouvrir

C'est exactement le même principe que BarreOption, elle hérite de ActionListener, possède des JButton et la méthode ActionPerformed(ActionEvent e) qui va agir sur le modèle grâce au contrôleur.

La seule différence réside sur l'héritage principal de la classe. En effet Celle-ci n'hérite pas de JPanel mais de JToolBar. L'objet JToolBar va permettre à l'utilisateur de décrocher la barre de la fenêtre pour obtenir une barre indépendante dans une nouvelle fenêtre. Il suffira de fermer cette nouvelle fenêtre pour que la barre se remette à sa position initiale.

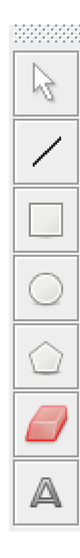


FIGURE 4.4 – Barre d'outils

Classe BarreEtatHaut

C'est toujours le même principe que BarreOption, elle hérite de ActionListener, possède des JButton (pour mettre en gras ou souligné) et la méthode ActionPerformed(ActionEvent e) qui va agir sur le model grâce au contrôleur.

Mais ce n'est pas tout. Cette classe possède des JSpinner (objet Swing) qui sont des boites de texte avec des flèches (vers le haut et le bas) qui vont permettre de modifier la valeur de la boite correspondante.

Pour savoir quand l'utilisateur modifie ces objets, la classe BarreEtatHaut doit hériter de l'interface ChangeListener, celle-ci va permettre d'utiliser la méthode stateChanged(ChangeEvent e) qui sera appelée dès qu'un JSpinner sera modifié. Bien sûr, le contrôleur sera utilisé dans cette méthode pour faire la liaison avec le model.

Pour gérer les évènements (pour annuler/rétablir), la classe hérite de FocusListener. Celle-ci va permettre de créer un évènement (grâce au contrôleur) au moment où il clique sur un des JSpinner (c'est le moment ou la barre gagne le focus) grâce aux méthodes focusGained(FocusEvent e) et focusLost(FocusEvent e).

Cette classe possède aussi des JLabel. Ce sont des objets qui servent à afficher du texte ou des images assez facilement. Dans notre cas on les utilise pour afficher du texte. Il y a aussi une JList qui permet d'afficher la liste des polices.

On distinguera ensuite quatre skins différents pour cette barre :

– Skin vide : quand aucun objet n'est sélectionné.



Figure 4.5 – Barre d'outils vide

– Skin pour forme normal (cercle, rectangle, ligne) : Permet de modifier l'opacité et la largeur de la bordure d'une figure.

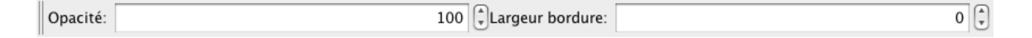


Figure 4.6 – Barre d'outils pour un cercle, un rectangle et une ligne

– Skin pour polygone : on ajoute le nombre de côtés du polygone au skin pour forme normal.



Figure 4.7 – Barre d'outils pour le polygone

– Skin pour Texte : permet de modifier la police, l'opacité et mettre en gras, souligné.

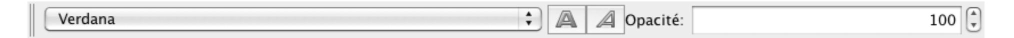


FIGURE 4.8 – Barre d'outils pour le texte

Les méthodes permettant de modifier le skin de la barre ne sont pas décrites ici car elles sont déjà commentées dans le code.

Classe BarreEtatBas

Celle-ci possède que des JLabel. Ici ils servent à afficher du texte ou des carrés de couleur. Cette classe hérite de JPanel pour disposer facilement les objets qui la composent.

Elle hérite aussi de MouseListener qui va permettre d'écouter les évènements liées à la souris. Ce sera donc la méthode mouseClicked(MouseEvent e) qui se chargera de voir si on a cliqué sur l'un des carrés de couleur pour pouvoir ensuite afficher la fenêtre de choix de couleurs qui est une JColorChooser de la libraire Swing.

On peut voir en bas à droite de la barre, des coordonnés (X, Y), ils représentent les coordonnées de la souris dans la zone de dessin. Ils sont rafraichis grâce à une méthode du contrôleur appelée par une méthode de la zone de dessin.



FIGURE 4.9 – Barre d'état

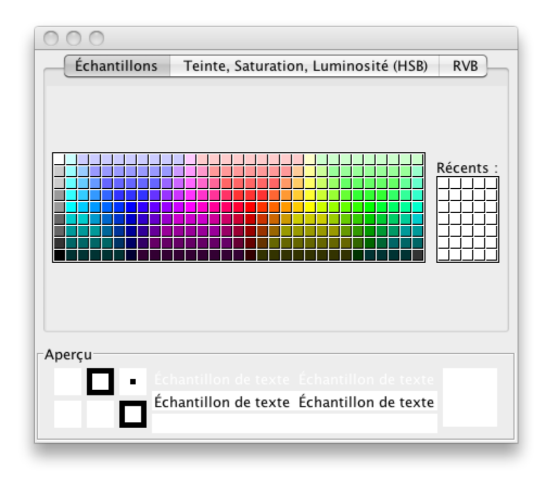


FIGURE 4.10 – Fenêtre pour changer de couleur

Classe ZoneDeDessin

Celle-ci hérite de JPanel car les JPanel sont les objets les plus simples à utiliser de la librairie Swing pour dessiner. Le JPanel possède un objet Graphics2D g qui possède diverses méthodes permettant de dessiner tous types de formes de bases :
- g.drawRect pour dessiner des rectangles
- g.drawOval pour dessiner des ellipses
- g.drawPolygon pour dessiner des poylgones
- g.drawLine pour dessiner des lignes
- g.drawString pour dessiner du texte.

On a ensuite des méthodes pour dessiner le reste des propriétés des objets :
- g.setColor pour donner la couleur du dessin
- g.setStroke pour donner l'épaisseur de la bordure du dessin (zéro si pas de bordure)
- g.setComposite pour donner la valeur de l'opacité du dessin
- g.setFont pour donner la feuille de style

Grâce à cet objet Graphics2D nous allons pouvoir par la suite afficher les motifs qui composent le dessin vectoriel.

Il faut ensuite gérer les évènements de la souris. La classe héritera donc de MouseListener et de MouseMotionListener.

MouseListener va permettre de gérer tout ce qui est en rapport avec le clic :
- le clic de la souris : mouseClicked(MouseEvent e)
- la pression sur le bouton gauche de la souris : mousePressed(MouseEvent e)
- le relâchement du bouton gauche de la souris : mouseReleased(MouseEvent e)

MouseMotionListener va permettre de détecter les mouvements :
- Quand la souris est en mouvement : mouseMoved(MouseEvent e)
- Quand la souris est en mouvement et que le clic gauche est enfoncé mouseDragged(MouseEvent e)

Tous ces écouteurs utiliseront ensuite le contrôleur et l'objet Graphics2D pour pouvoir dessiner à l'écran les objets qui composent l'image vectorielle.

4.3 Le contrôleur

C'est la classe la plus grosse du projet. C'est elle qui établit la liaison entre la vue et le modèle. Elle possède une multitude de fonctions qui vont elles-mêmes appeler d'autres fonctions du modèle ou de la vue ou les deux pour assurer l'indépendance des deux packages.

Lors de sa création, au moment où le constructeur est appelé, il va créer une instance du model qui sera l'objet dessin de type Dessin et une instance de la vue qui sera l'objet fenêtre de type Fenetre. Pour respecter le model MVC, le contrôleur est passé en paramètre au constructeur de la vue. Cela permettra à la vue, lors des différentes actions de l'utilisateur, de modifier le modèle comme il se doit pour que l'image vectorielle prenne forme.

Les actions de l'utilisateur sont gérées par des écouteurs (ou Listener en Java), contenues dans la vue, qui vont appeler les méthodes correspondantes du contrôleur.

Chapitre 5

Manuel d'utilisation

Le logiciel DVect est une application permettant de réaliser des dessins vectoriels. Tout au long de ce manuel nous allons voir comment utiliser toutes les fonctionnalitées de cette application pour avoir une maitrise totale du logiciel.

5.1 Interface graphique

L'interface graphique se divise en plusieur partie, nous allons découvrir ces différentes parties.

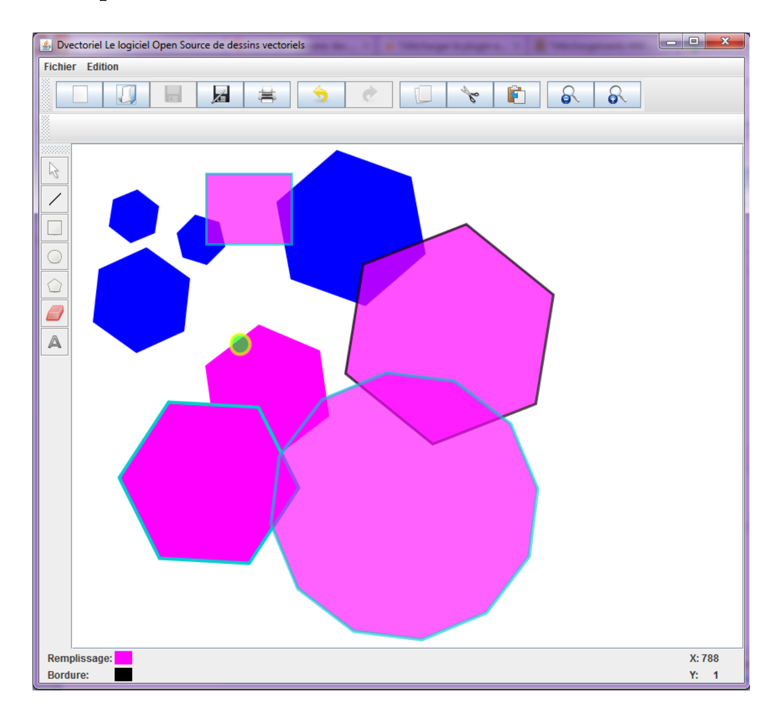


FIGURE 5.1 – Interface graphique

Nous allons découper notre interface graphique en six parties distinctes :

1. La **barre de menu déroulant** comporte les menus déroulants relatifs à l'application.
2. La **barre d'options** donne accès à des boutons permettant d'effectuer de nombreuses opérations sur le dessin actif.
3. La **barre d'édition** permet de modifier les caractéristiques du motif courant.
4. La **barre d'outils** contient une serie de bouton permettant de choisir le motif que vous voulez dessiner sur le dessin.
5. La **barre d'état** contient différentes informations relatives au dessin.
6. La **zone de dessin** permet d'afficher le dessin courant réalisé par l'utilisateur.

Barre de menu déroulant

La barre de menu déroulant c'est là où il y va avoir les différentes actions qui pourrons être exécutées sur le dessin. Cette barre de menu est constituée de deux menu : **Fichier** et **Edition**.



FIGURE 5.2 – Barre de menu déroulant

Fichier

Le sous-menu **Fichier** contient tous les fonctions qui nécessitent l'utilisation de fichiers sur le disque.



FIGURE 5.3 – Sous-menu **Fichier**

- L'entrée **Nouveau** permet de créer un nouveau dessin. Cela efface l'ancien dessin (si il y en a un) et crée un nouveau dessin sans nom et de taille 800 pixels de largeur par 600 pixels de hauteur.
- L'entrée **Ouvrir** permet de charger un dessin au format SVG depuis un fichier.
- L'entrée **Enregistrer** permet de sauvegarder le dessin actif sous son nom d'origine. Si aucun nom n'a été préalablement donné au document, le sélecteur de fichier « Enregistrer sous » apparaîtra pour saisir le nom du fichier.
- L'entrée **Enregistrer sous** permet de sauvegarder le dessin actif sous un autre nom que celui d'origine. Elle appelle le sélecteur de fichier « Enregistrer sous ».
- L'entrée **Exporter** permet d'exporter un dessin en divers formats (PNG, ...).
- L'entrée **Imprimer** permet d'imprimer le dessin actif. Elle appelle la boîte de dialogue « Impression ».
- L'entrée **Quitter** permet de quitter l'application DVect. Si le dessin n'a pas été sauvés, une boîte de dialogue s'ouvrira afin de proposer leur sauvegarde.

Edition

Le sous-menu **Edition** donne accès aux fonctions de communication avec le presse-papiers et avec l'historique d'évènement effectuée sur le dessin.

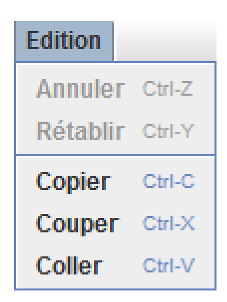


FIGURE 5.4 – Sous-menu **Edition**

- L'entrée **Annuler** permet d'annuler une action précédemment executée sur un motif précis.
- L'entrée **Rétablir** permet de rétablir une action précédemment annulée par l'utilisateur.
- L'entrée **Couper** permet de couper le motif et de le copier dans le presse-papiers. Elle agit de la même manière que le troisième bouton icône de la troisième section de la barre d'options.
- L'entrée **Copier** permet de copier le motif dans le presse-papiers. Elle agit de la même manière que le premier bouton icône de la troisième

section de la barre d'options.
– L'entrée **Coller** permet de coller le motif dans le dessin. Elle agit de la même manière que le deuxième bouton icône de la troisième section de la barre d'options.

Barre d'options

La barre d'option donne accès à des boutons icônes permettant d'effectuer de nombreuses opérations sur le dessin. Il y a notamment le bouton nouveau, ouvrir, enregistrer, enregistrer sous, imprimer, annuler, rétablir, copier, coller, couper, zoomer et anfin dézoomer.



FIGURE 5.5 – Barre d'options

1. Ce bouton permet de créer un nouveau dessin. Cela efface l'ancien dessin (si il y en a un) et crée un nouveau dessin sans nom et de taille 800 pixels de largeur par 600 pixels de hauteur.
2. Ce bouton permet d'ouvir un dessin au format SVG .
3. Ce bouton permet d'enregistrer le dessin actif sous son nom d'origine. Si aucun nom n'a été préalablement donné au dessin, le sélecteur de fichier « Enregistrer sous » apparaîtra pour saisir le nom du fichier.
4. Ce bouton permet de sauvegarder le dessin actif sous un autre nom que celui d'origine. Elle appelle le sélecteur de fichier « Enregistrer sous ».
5. Ce bouton permet d'imprimer le dessin actif. Elle appelle la boîte de dialogue « Impression ».
6. Ce bouton permet d'annuler une action précédemment executée sur un motif précis.
7. Ce bouton permet de rétablir une action précédemment annulée par l'utilisateur.
8. Ce bouton permet de couper le motif et de le copier dans le presse-papiers.
9. Ce bouton permet de copier le motif dans le presse-papiers.
10. Ce bouton permet de coller le motif dans le dessin.
11. Ce bouton permet de dézoomer sur le dessin.
12. Ce bouton permet de zoomer sur le dessin.

Barre d'édition

La barre d'édition permet de modifier les différentes caractéristiques du motif courant. Selon le motif courant, ses caractéristiques ne sont pas les mêmes c'est pour cela que cette barre peut avoir différents champs.

Opacité



Figure 5.6 – Opacité

Ce champ permet de modifier l'opacité d'un motif. L'opacité d'un motif est comprise entre 0% (invisible) et 100% (opaque), par défaut un motif a une opacité égale à 100%.

Largeur bordure



Figure 5.7 – Largeur bordure

Ce champ permet de modifier la largeur de la bordure d'un motif. La largeur d'un motif varit entre 0 pixel et 255 pixels, par défaut un motif a une largeur de bordure égale à 0 pixel.

Nombre de côtés



Figure 5.8 – Nombre de côtés

Ce champ permet de modifier le nombre de côtés d'un motif de type polygone. Le nombre de côtés d'un polygone varit de 4 à 512, par défaut un polygone a 6 côtés et le nombre de côtés d'un polygone est forcément pair.

Police du texte

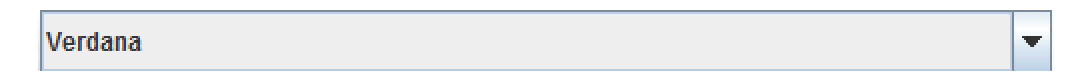


Figure 5.9 – Police du texte

Ce champ permet de modifier la police d'un motif de type texte. Par défaut un texte sa police est égale à « Verdana ».



FIGURE 5.10 – Gras

Gras

Ce bouton permet de mettre ou d'enlever la propriété gras à un texte. Par défaut un texte n'est pas en gras.

Italique



FIGURE 5.11 – Italique

Ce bouton permet de mettre ou d'enlever la propriété italique à un texte. Par défaut un texte n'est pas en italique.

Barre d'outils

La barre d'outils contient une serie de bouton permettant de choisir le motif que vous voulez dessiner sur le dessin. L'utilisateur peut choisir l'outil souris, ligne, carré, ellipse, polygone, gomme, texte.



FIGURE 5.12 – Barre d'outils

1. Ce bouton change l'outil courant avec l'outils « souris » ce qui permet de sélectionner, agrandir ou de déplacer un motif du dessin.

2. Ce bouton change l'outil courant avec l'outils « ligne » ce qui permettra de dessiner une ligne.
3. Ce bouton change l'outil courant avec l'outils « carré » ce qui permettra de dessiner un carré.
4. Ce bouton change l'outil courant avec l'outils « ellipse » ce qui permettra de dessiner une ellipse.
5. Ce bouton change l'outil courant avec l'outils « polygone » ce qui permettra de dessiner un polygone.
6. Ce bouton change l'outil courant avec l'outils « gomme » ce qui permettra d'effacer un motif de dessin.
7. Ce bouton change l'outil courant avec l'outils « texte » ce qui permettra d'écrire du texte dans le dessin.

Barre d'état

Cette barre permet d'afficher tous les informations courantes du dessin. Notamment la position de la souris, la couleur de remplissage d'un motif et la couleur de sa bordure.

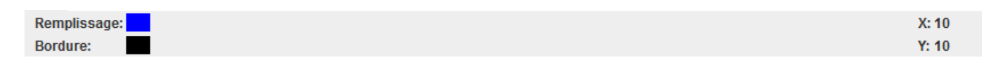


FIGURE 5.13 – Barre d'état

Zone de dessin

La zone de dessin permet d'afficher le dessin courant réalisé par l'utilisateur. C'est dans cette zone que vous pourrez intéragir avec le dessin.



FIGURE 5.14 – Zone de dessin

5.2 Les actions

Dans cette partie nous verrons toutes les actions possibles à executer avec cette application. Nous expliquerons comment accéder à toutes ces actions et nous verrons comment les réaliser.

Interface

Comme nous l'avons vu, l'interface graphique se divise en plusieurs parties.

Nouveau dessin

Fichier → Nouveau
Racourci : ctrl + n
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Ouvrir dessin

Fichier → Ouvrir
Racourci : ctrl + o

Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Enregistrer

Fichier → Enregistrer
Racourci : ctrl + s
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Enregistrer sous

Fichier → Enregistrer sous
Racourci : ctrl + maj + s
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Impression

Fichier → Impression
Racourci : ctrl + p
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Annuler

Edition → Annuler
Racourci : ctrl + z
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Rétablir

Edition → Rétablir
Racourci : ctrl + y
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Copier

Edition → Copier
Racourci : ctrl + c
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Couper

Edition → Couper
Racourci : ctrl + x
Vous pouvez aussi y accéder par la **barre d'option** (voir le chapitre précèdent).

Coller

Edition → Coller
Racourci : ctrl + v
Vous pouvez accéder par la **barre d'option** (voir le chapitre précèdent).

Dézoomer

Vous pouvez y accéder par la **barre d'option** (voir le chapitre précèdent).

Zoomer

Vous pouvez y accéder par la **barre d'option** (voir le chapitre précèdent).

Changer les caractéristiques d'un motif

Vous pouvez y accéder par la **barre d'édition** (voir le chapitre précèdent).

Changer la couleur

Vous pouvez accéder à la couleur grâce à la **barre d'état**, il vous suffit de cliquer sur la couleur pour q'une fenêtre permette de changer la couleur.

Changer d'outils

Vous pouvez aussi y accéder par la **barre d'outils** (voir le chapitre précédent).

Motif

Nous allons aborder toutes les intéractions que l'on peut réaliser sur un motif (de sa création à sa suppression).

Création

Pour la création du motif, tout d'abord il vous faut choisir le motif que vous désirez dessiner. Ensuite selon le motif que vous avez choisi, la manière de le dessiner peut être différente.

- Pour dessiner une « ligne » dans la zone de dessin, vous devez effectuer un clique gauche pour déterminer le premier point de la ligne et ensuite ne pas relacher le bouton de la souris pour déterminer le deuxième point de la ligne en relachant se dernier.
- Pour la création d'un « rectangle » ou d'une « ellipse » dans la zone de dessin, vous devez procéder exactement de la même manière que pour la ligne, en revanche les deux points qui auront été déterminé seront la diagonale de l'ellipse ou du rectangle.
- Pour dessiner un « polygone » dans la zone de dessin, c'est exactement comme le rectangle et l'ellipse par contre vous pouvez tourner le polygone dans la direction que vous voulez, pour comprendre comment le faire tourner regarder cette explication :

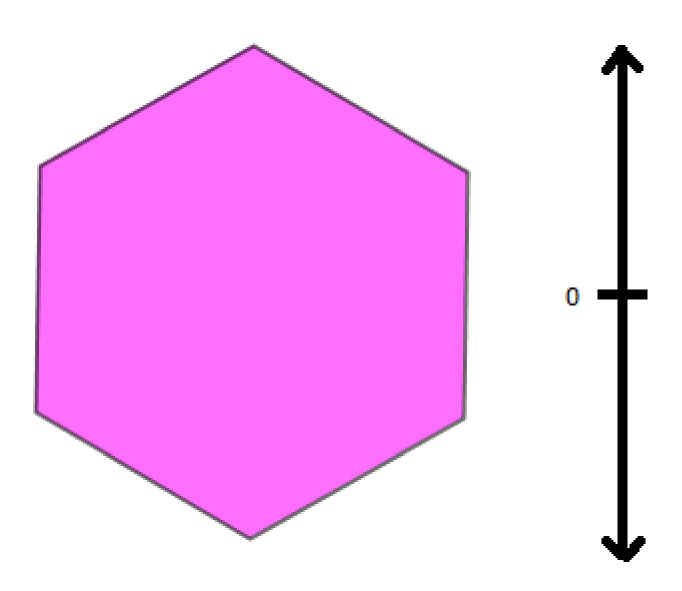


Figure 5.15 – Explication rotation polygone

La rotation du polygone s'effectue en fonction de l'axe y, si votre souris est à la même coordonnée y que la coordonnée y du centre du polygone alors le polygone sera droit, sinon le polygone sera tourné en fonction du décalage de la souris sur l'axe y avec le centre du polygone.

- Enfin pour écrire du « texte » dans la zone de dessin, il vous suffit de

cliquer là où vous voulez le placer et ensuite une fenêtre s'ouvrira pour écrire le texte que vous désirez dessiner dans la zone de dessin.

Modification

Pour modifier un motif, il vous suffira de prendre l'outil « souris » et de cliquer sur le motif que vous voulez modifier, ensuite la « barre d'édition » se modifira en fonction du motif choisi ce qui vous permettra de modifier certaine caractéristique du motif (voir le chapitre sur la barre d'édition). Mais vous pouvez aussi agrandir le motif grâce aux poignées qui se seront affichées autour du motif. Enfin vous pouvez aussi le déplacer en effectuant un clic gauche dessus sans le relacher, puis de bouger la souris là ou vous voulez mettre le motif. Une fois le motif à la place que vous désirez, il vous suffit de relacher le clic gauche.

Suppression

Pour supprimer un motif, il vous suffit de prendre l'outils « gomme » et ensuite il vous reste plus qu'à faire un clic gauche sur le motif que vous voulez supprimer et le motif sera supprimé du dessin.

Chapitre 6

Perspectives et discussions

6.1 Perspectives

Le projet fonctionne correctement mais nous pourrions implémenter des fonctionnalités supplémentaires qui pourraient fortement enrichir nos connaissances et améliorer le projet. L'implémentation de calque serait très utile pour l'utilisateur et n'est pas difficile à implémenter grâce à la structure de notre projet. Un autre ajout utile serait d'implémenter la rotation d'un motif, celle-ci devrait s'implémenter facilement grâce à un objet de la librairie Swing conçu pour cet effet. Ces améliorations devraient permettre à un utilisateur débutant dans le domaine du dessin vectoriel de profiter pleinement de l'application.

6.2 Discussion

Il faut savoir que ce logiciel de dessin vectoriel est une version simplifiée des logiciels existant sur le marché, c'est pour cela qu'il reste encore beaucoup d'amélioration à implémenter pour qu'il soit utilisable par des personnes plus expérimenté dans ce domaine. Pour finir on peut dire que le projet que nous avons réalisé cette année est destiné à évoluer pour sortir du cadre scolaire.

Chapitre 7

Conclusions

7.1 Fonctionnement de l'application

L'application que nous avons réalisée est fonctionnelle et respecte les contraintes de fonctionnalités minimales imposées.

Durant le développement et les tests que nous avons réalisé, nous avons décelé quelques bogues minimes. Ceux-ci ont été corrigés au jour d'aujourd'hui.

Toutes les fonctionnalités implémentées fonctionnent parfaitement, du dessin de l'objet à l'enregistrement de celui-ci.

Le choix du langage a été fructueux. Java est très intuitif et facile à apprendre quand on connait le concept de programmation par objet. La librairie Swing a été très utile et nous a permis de réaliser facilement l'interface graphique.

Quant à l'analyse du projet. Elle s'est avéré suffisante et efficace car l'application fonctionne et que l'implémentation d'une multitude de fonctionnalités est envisageable.

7.2 Fonctionnement du groupe de travail

Nous avons du développé cette application en binôme. Cela a été un handicap car nous disposions d'un temps limité et que le développement d'une application comme celle-ci demande une vraie équipe de projet composée de plusieurs membres. Heureusement, grâce à notre organisation nous avons su gérer le temps. Notre petit effectif nous a permis d'établir une très bonne entente et une facilité à communiquer qui nous a permis de programmer efficacement, de nous entraider tout au long de ce projet et surtout de partager nos connaissances.

Chapitre 8

Annexe

API

Pour l'API de l'application de vous laisse la télécharger sur http ://code.google.com/-p/dvectoriel/downloads/list.

Digrammes UML

Comme pour l'API, vous pouvez télécharger tous les diagrammes UML sur http ://code.google.com/p/dvectoriel/downloads/list.

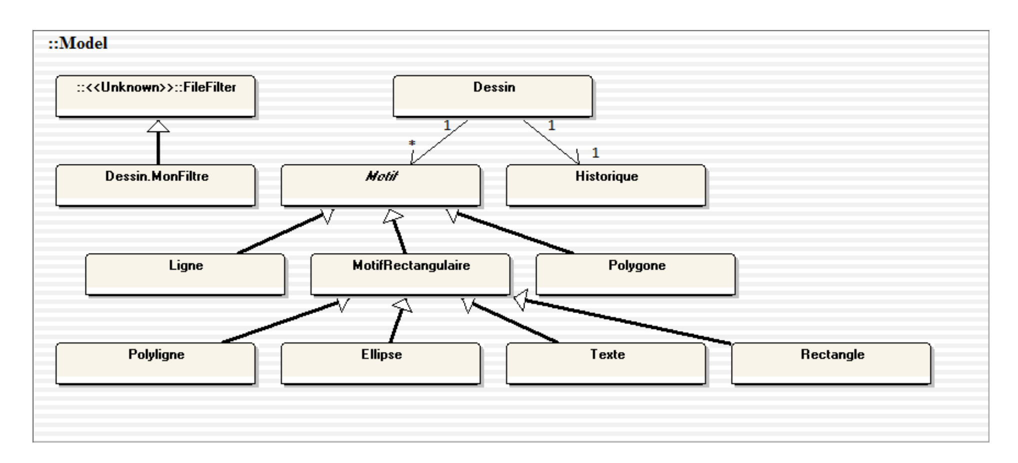


FIGURE 8.1 – Diagramme UML du modèle (simplifié)

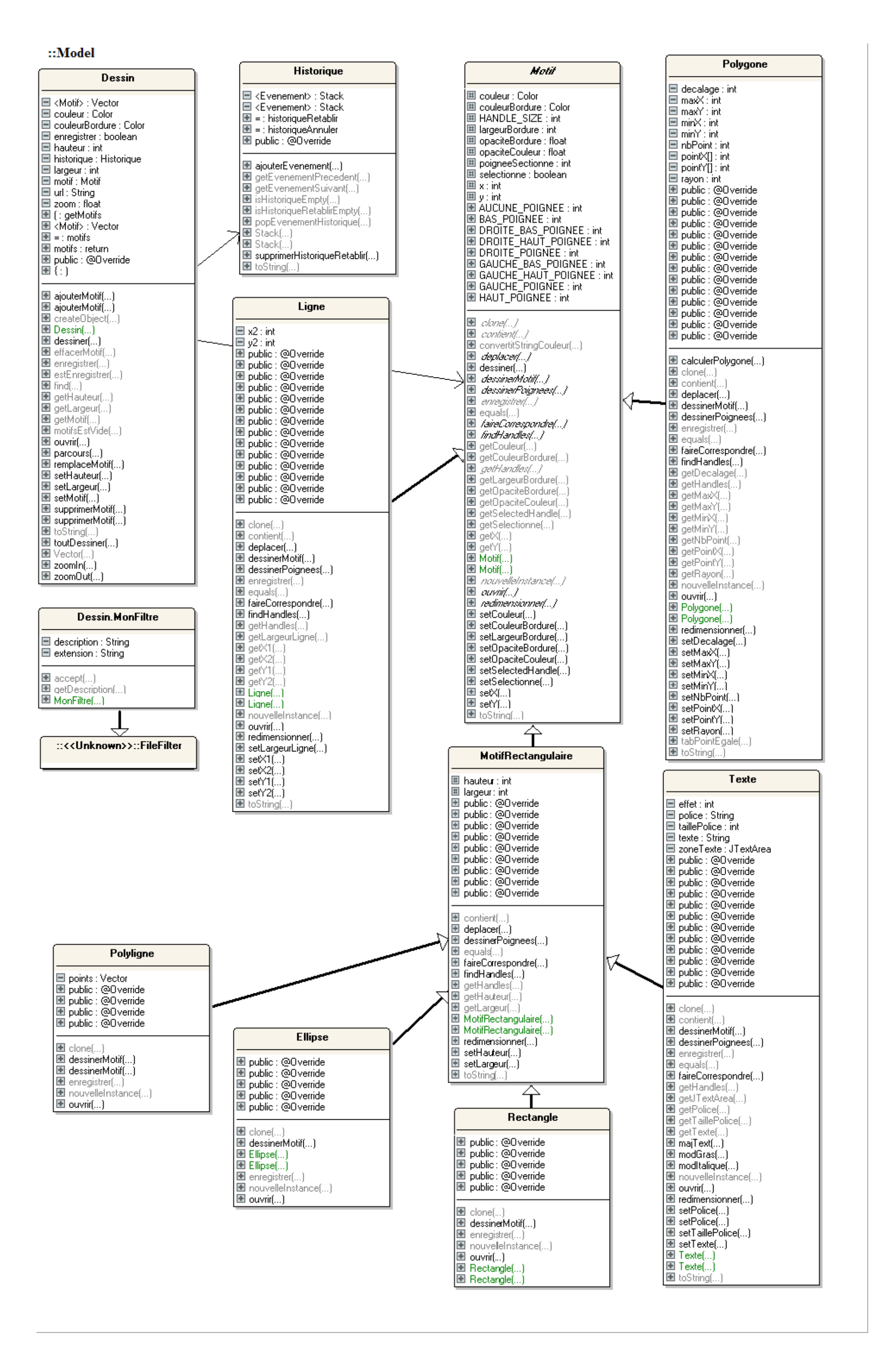


Figure 8.2 – Diagramme UML du modèle

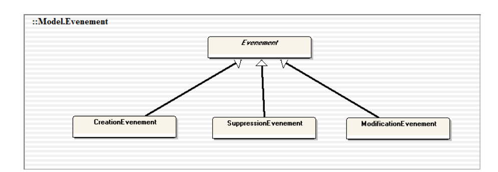


FIGURE 8.3 – Diagramme UML des évènements du modèle (simplifié)

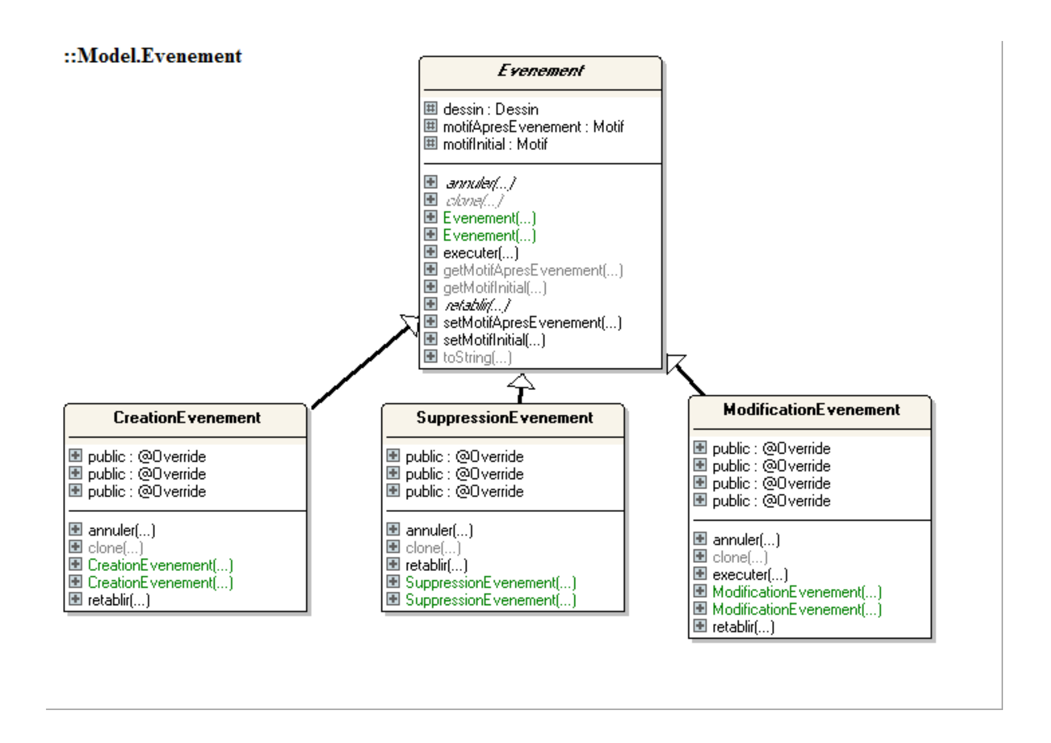


FIGURE 8.4 – Diagramme UML des évènements du modèle

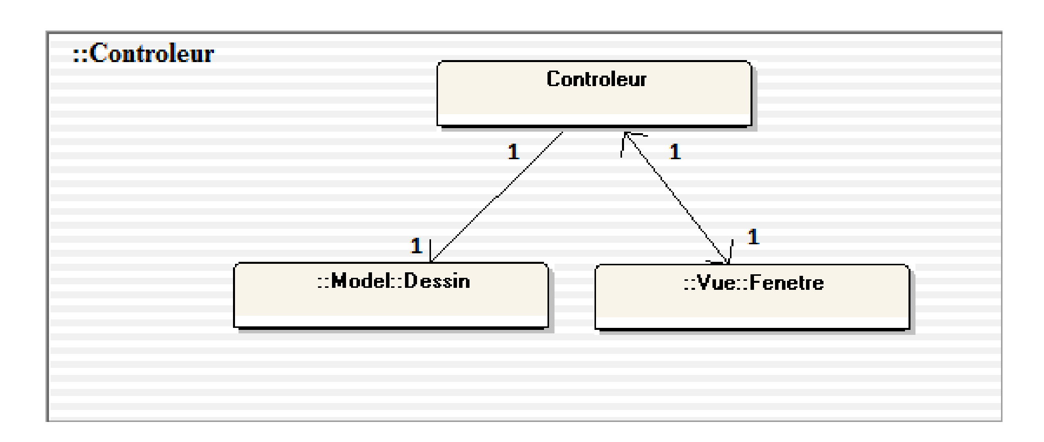


FIGURE 8.5 – Diagramme UML du contrôleur (simplifié)

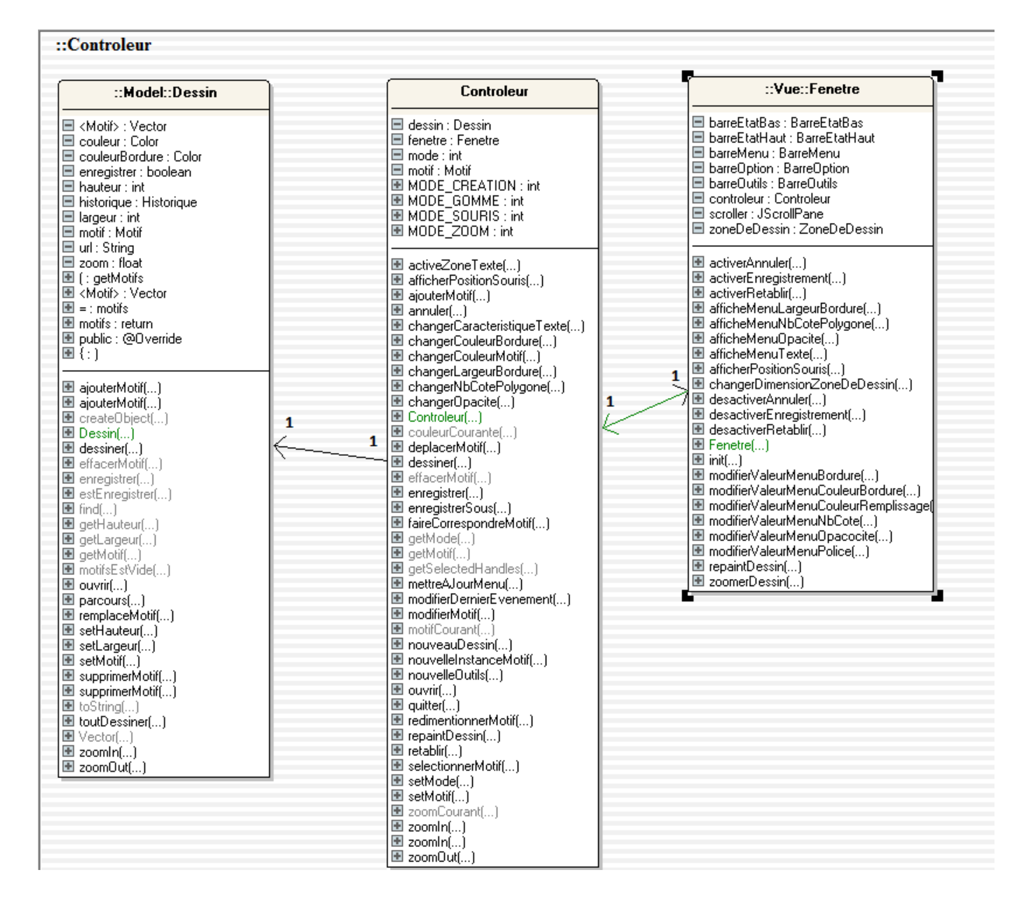


FIGURE 8.6 – Diagramme UML du contrôleur

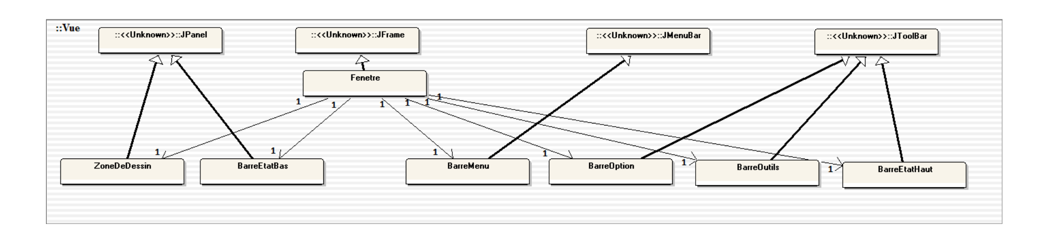


FIGURE 8.7 – Diagramme UML de la vue (simplifié)

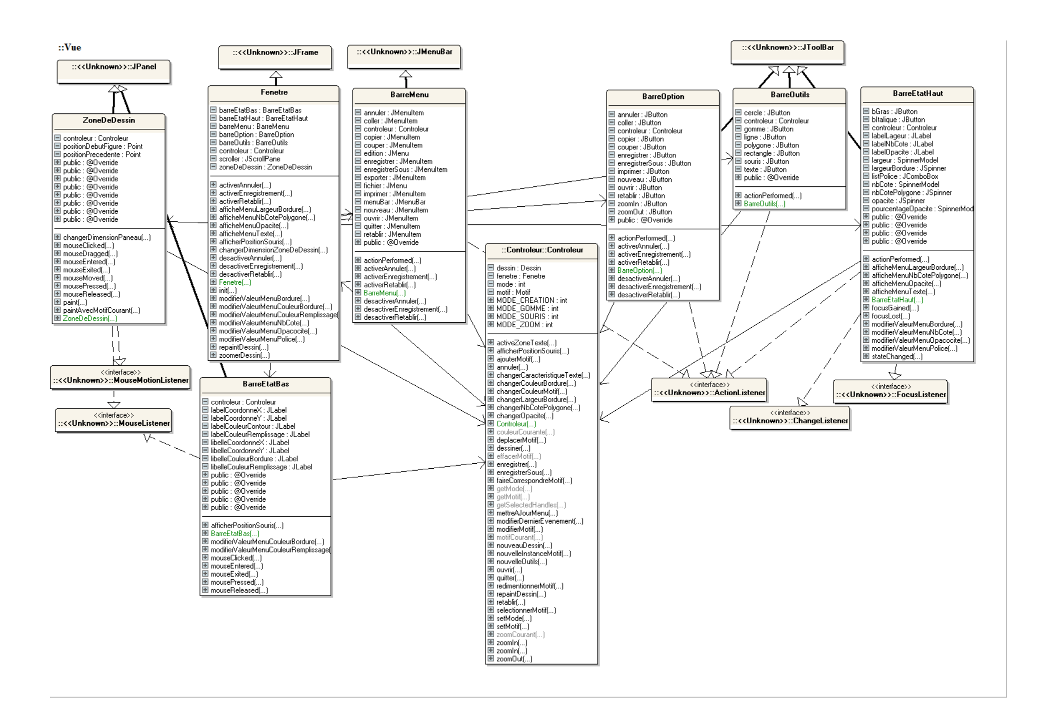


FIGURE 8.8 – Diagramme UML de la vue